Processor Prototyping Lab Midterm Report

ECE 43700

Group Members: Christopher Priebe, Victor Le

Lab Section: 5

GTA: Vaibhav Ramachandran

Due Date: March 14, 2021

1 Overview

The basis of this report is to compare the performance and design of our single cycle and pipeline processor designs. We standardized our design by implementing the MIPS ISA for our test programs as well as using the same programs when comparing our designs. This normalizes comparisons and validate units of comparison such as MIPS (Millions of Instructions per Second). Our pipeline processor is a 5 stage design that ideally increases throughput by simultaneously running multiple instructions in different stages. When designing the pipeline processor, we encountered hazards, or issues, when reading data the has not fully been executed within our pipeline stages. In order to counteract these issue, we designed sub-units within our datapath in order to produce a functional processor without stalling. The three modules we designed include the hazard unit, the forwarding unit, and the branch prediction unit. The hazard unit's original intention was to detect data hazards and stall whenever the hazard is detected so that the pipeline has the opportunity to write back to registers. The hazard unit also originally detected branch taken conditions which would flush the FE, DEC, and EX stages in order to correctly determine the data that should be passed through the pipeline. The second module we implemented was the forwarding unit which would utilized the information gather by the hazard unit in order to forward updated data used by future stages. The final module we implemented was the branch prediction module which either uses a 2-bit saturation predictor or a branch target buffer. The intention of this module was to intuitively predict the next branch instruction, which would reduce the amount of occurrences being penalize by a missed branch prediction.

When comparing the respective designs of each processor, we used the given mergesort.asm file which uses a mergesorting algorithm that would be complex enough to illustrate the differences in performance between the two designs. Some additionally metrics of measurements include the frequency and instruction latency given that we used CPI and MIPS. All of these measurements were done from a latency of 0 to 10 in increments of 2. When analyzing the data we noticed that the frequency increased by 1.75 time from our single cycle design to our pipeline design.

2 Processor Design

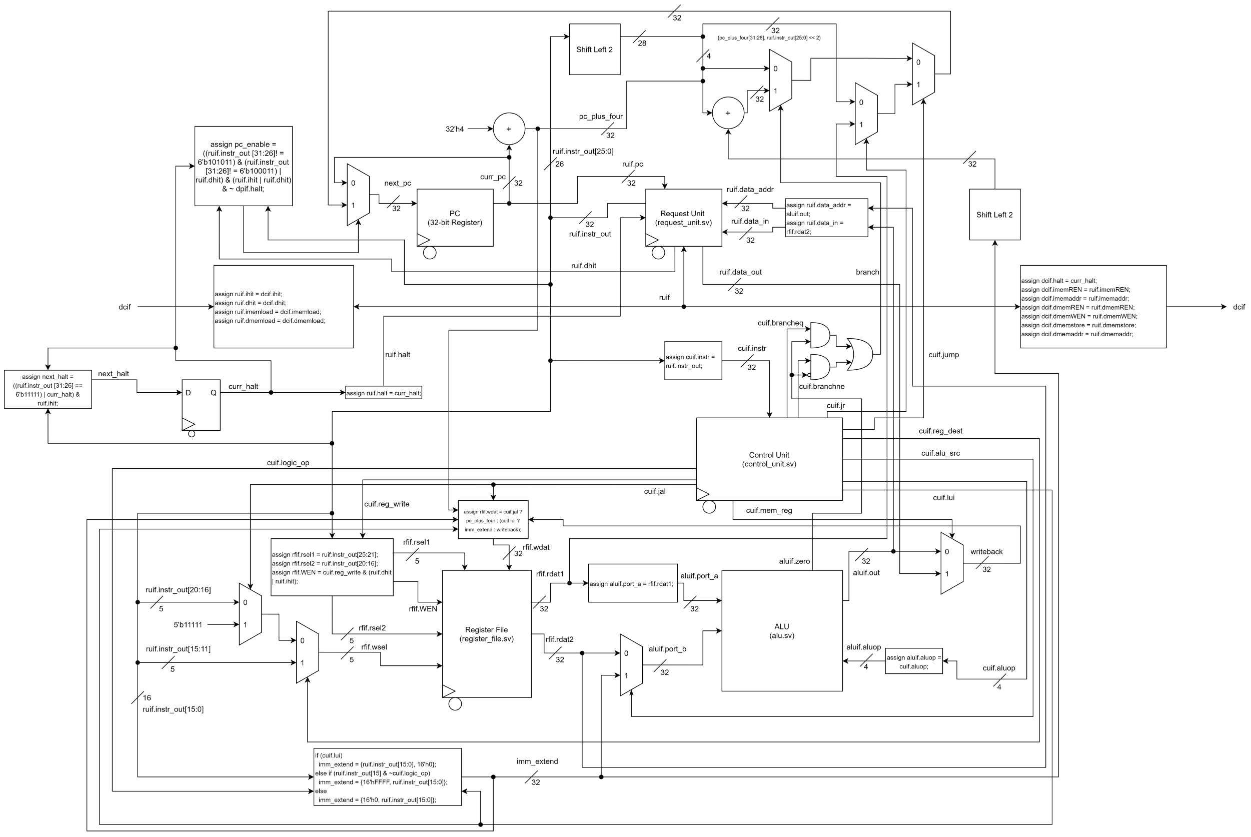
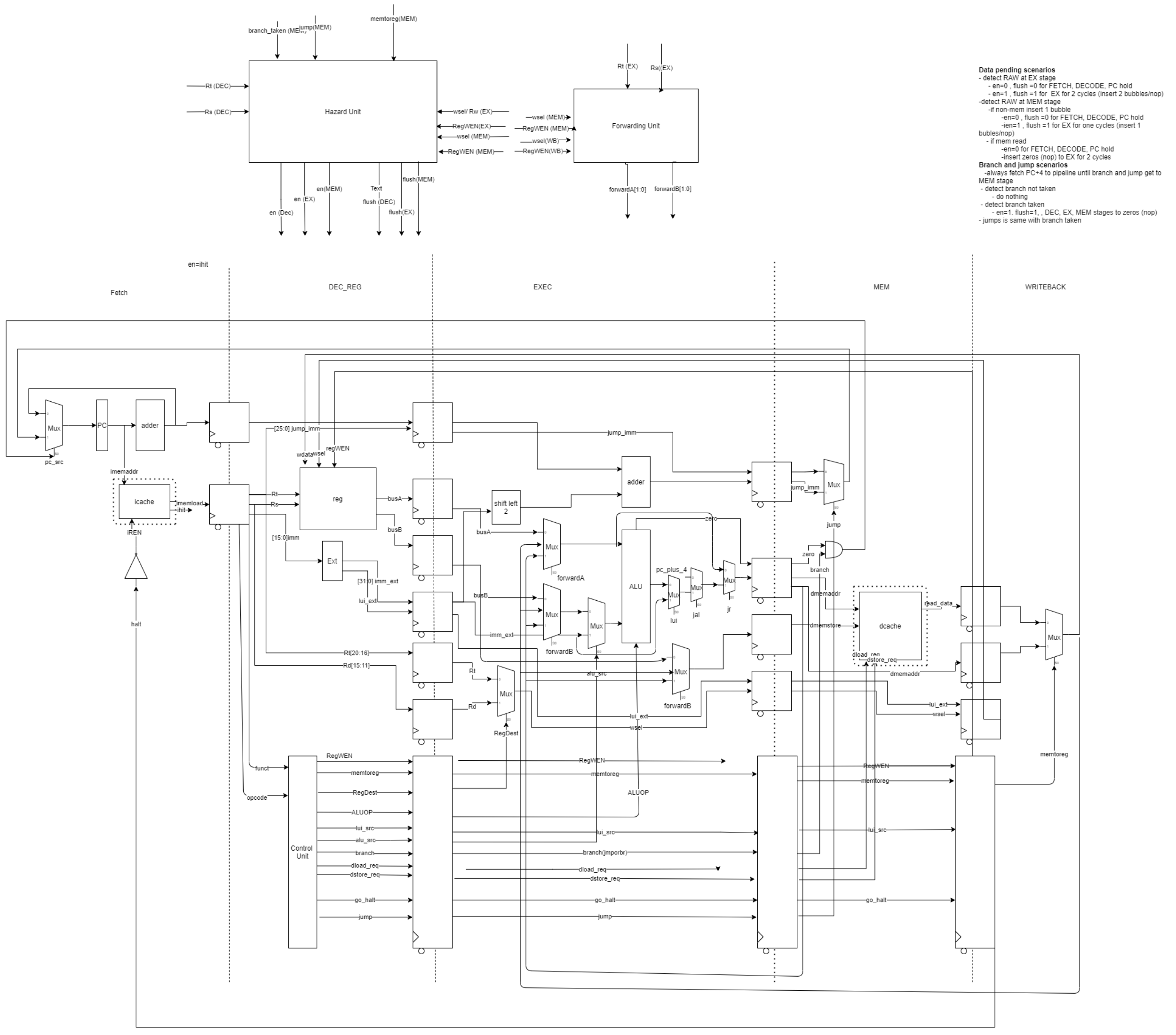


Figure 1: The single cycle processor block diagram.

The RTL block diagram for the single cycle processor is shown in Figure 1. This processor is a basic implementation of the MIPS ISA, supporting basic R-type instructions, arithmetic and logical immediate instructions, and the following control instructions: jump, jump and link, jump register, branch equal, and branch not equal. The RTL block diagram for the pipelined processor is shown in Figure 2. This processor implements the same instructions as its single cycle counterpart, but also implements a 5-state pipeline with full control hazard detection and stalling, full RAW hazard forwarding, and a branch-always-taken predictior for branch instructions.



Data pending scenarios
- detect RAW at EX stage
    - en=0 , flush =0 for FETCH, DECODE, PC hold
    - en=1 , flush =1 for EX for 2 cycles (insert 2 bubbles/nop)
-detect RAW at MEM stage
    -if non-mem insert 1 bubble
        -en=0 , flush =0 for FETCH, DECODE, PC hold
        -ien=1 , flush =1 for EX for one cycles (insert 1 bubles/nop)
    - if mem read
        -en=0 for FETCH, DECODE, PC hold
        -insert zeros (nop) to EX for 2 cycles

Branch and jump scenarios
-always fetch PC+4 to pipeline until branch and jump get to MEM stage
- detect branch not taken
    - do nothing
- detect branch taken
    - en=1. flush=1, , DEC, EX, MEM stages to zeros (nop)
- jumps is same with branch taken

Figure 2: The pipelined processor block diagram.

3 Results

The results shown in Table 1 shows the data collected for the single cycle processor design shown Figure 1 based on its performance with a program implementing merge sort. The results shown in Table 2 shows the data collected for the pipelined processor design shown Figure 2 based on its performance with a program implementing merge sort. The maximum frequency was found in the log file produced during the synthesis of the processors RTL code.

Metric	LAT: 0	LAT: 2	LAT: 4	LAT: 6	LAT: 8	LAT: 10
Max Frequency (MHz)	41.89	40.62	41.28	39.18	38.47	39.77
CPI ($^{\text{instr}}/_{\text{cycles}}$)	1.28	2.55	3.83	5.11	6.39	7.66
Instruction Latency ($^{\text{instr}}/_{\text{ns}}$)	51.80	102.17	153.25	204.33	255.41	306.50
MIPS	19.58	9.79	6.53	4.89	3.92	3.26
Total Registers	1314	1318	1318	1318	1318	1318
Total Comb. Functions	2,869	2,881	2,879	2,881	2,890	2,881

Table 1: The single cycle processor specifications.

Metric	LAT: 0	LAT: 2	LAT: 4	LAT: 6	LAT: 8	LAT: 10
Max Frequency (MHz)	71.6	69.65	72.96	69.96	75.79	68.98
CPI ($^{\text{instr}}/_{\text{cycles}}$)	2.42	3.7	5.5	6.8	8.1	9.37
Instruction Latency ($^{\text{instr}}/_{\text{ns}}$)	97	148	220	272	323	373
MIPS	10.3	6.76	4.54	3.68	3.09	2.6
Total Registers	1672	1672	1672	1672	1672	1672
Total Comb. Functions	3,404	3,402	3,404	3,400	3,401	3,405

Table 2: The pipelined processor specifications.

The merge sort program that both processors were tested with, when executed, consisted of 5,404 instructions. The CPI for each of the processors was calculated using the formula $\text{CPI} = \frac{\#\text{CPU\_cycles}}{5404}$. Both programs were simulated using a gate-level simulation at 50 MHz. The instruction latency was calculated using the formula $\text{latency} = \frac{\text{total\_time}}{5404}$. MIPS was calculated using the formula $\text{MIPS} = \frac{1}{\text{latency}} \cdot 10^{-6}$. The FPGA resources were found in the summary file produced during the synthesis of the mapped design to a representation of the mapped design on the lab FPGA's.

As seen in Table 1 and Table 2, the single cycle design performs better in every area except for maximum clock frequency. The reason the pipelined processor has a higher maximum clock frequency at all latencies is because of the shorter critical path between pipeline stages versus having the critical path being from the FE portion all the way to the

WB portion of the design. The pipelined processor should have had similar CPI results as the single cycle processor, as this is how the pipelined processor is implemented. However, there are a few key factors that increase the CPI of the pipelined processor. At low latencies, the CPI is dominated by the incorrect branch predictions made through the always-taken predictor. the merge sort algorithm has many branches, and therefore, there are many branches for the predictor to incorrectly predict. An incorrect prediction causes a three cycle stall because the branches are resolved in the MEM stage. At higher latencies, this is not as apparent and the CPI is closer to the memory latency because the larger amount of load and store instructions used in merge sort dominate the CPI. While the incorrect branch prediction stall of three cycles still exists, it is overshadowed by the 6+ cycle stall due to memory accesses at higher latencies. Other than this, all other results found are expected. Due to the multiple stage pipeline, instructions take longer to finish in a pipelined datapath. This causes the MIPS to decrease for a pipelined datapath as well. Because the pipelined datapath is more complicated than the single cycle datapath, the FPGA resource usage is greater. Overall, the pipelined processor performed well, and with improved branch prediction, would out perform the single cycle processor in CPI, causing it to be faster overall at similar workloads.

4 Conclusion

The pipeline design was intended to be more efficient than the single cycle design because the pipeline is design to execute more instruction cycles per cycle. However, in our design and our test we believe that there were sufficient amount of branch predictions missed and other stalls that could of created poor results for the pipeline design.

Overall, we had a worse CPI for the pipeline design, but it was able to run at a far greater frequency than the single cycle design. Interestingly, the pipeline even with this greater frequency fail to out perform the single cycle in run time, CPI, and MIPS. As expected there was a greater instruction latency for the pipeline design. In other test programs, we did fine to see that the pipeline design was overall more effective than the single cycle design.

5 Contributions

Both of our single cycle designs were fully functional, so we decided to keep working on Lab 5 individually while syncing up in order to debug any issues. By the end of Lab 6, we decided it would be more effective to have one person's design to implement the additional modules on. We went ahead with Chris' design because he had implemented the Hazard Unit and there were less issues at the system level. As a team we divided worked as we went and did not formally break down anything besides the design and test benches' of the forwarding and hazard units. As an overview, Chris contributed by designing the hazard unit, creating the test bench for the forwarding unit, integrating the hazard unit, debugging datapath and system level, and creating the bonus branch predictions. Whereas Victor contributed by designing the test bench for the hazard unit, designing the forwarding unit, implementing the forwarding unit, debugging latency issues, and creating block diagrams for all Labs. Individually we designed the originally pipeline without forwarding or hazard unit. As a team, during out meetings, we implemented always taken and always not taken branch predictions.